BINUS UNIVERSITY ONLINE LEARNING

People
Innovation
Excellence

LECTURE NOTES

Software Engineering

.

Minggu 4

Design Engineering

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☑ LO2 – Menjelaskan praktek *software engineering*

Outline Materi (Sub-Topic) :

1. *Architectural Design*
2. *Component-Level Design*
3. *User Interface Design*
4. *Pattern Based Design*
5. *WebApp Design*
6. *Mobile App Design*
7. Studi kasus

ISI MATERI

Proses desain atau design engineering dilakukan setelah fase *requirement engineering*. Proses ini juga sangat penting sebelum dilakukan proses implementasi atau *coding*. Pada proses pengembangan perangkat lunak dengan metode *agile*, proses desain ini dapat dilakukan pada setiap iterasi yang terjadi.

1. *Architectural Design*

Software architecture adalah sebuah desain umum suatu proses pada sebuah *software system*, meliputi:

- Pembagaian *software* ke dalam subsistem
- Memutuskan bagaimana saling berhubungan
- Menentukan alat penghubung

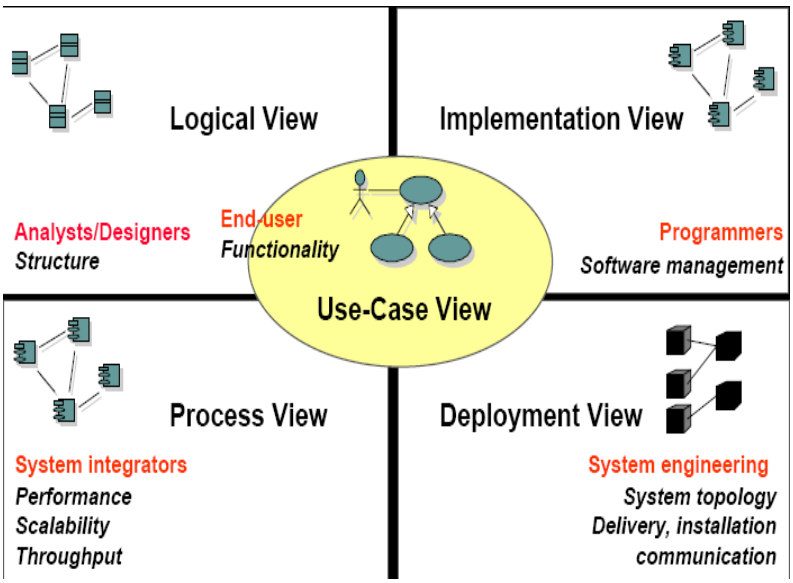
Pentingnya arsitektur sebuah *software*:

- Kenapa kita perlu mengembangkan arsitektur:
  - Agar setiap orang bisa mengerti mengenai sistem yang ada.
  - Untuk membiarkan *user* bekerja secara indivial terhadap sebuah sistem
  - Persiapan untuk perluasan *system*
  - Menyediakan fasilitas *reuse* and *reusability*

Struktur dan pandangan arsitektural:

- *View* menampilkan aspek aspek yang terdapat pada *software architecture* yang menunjukan spesifikasi *software*.
- *Architectural structures*
  - Sebuah sistem *family* yang terkait dengan *pattern*
  - sebuah *vocabulary* dari komponen dan *connector type*
  - Suatu batasan dimana dapat kombinasikan

Architectural structures dapat disebut juga dengan *architectural style*



- ***Architecture view***
  - ***Use Case View***
    - Analisa *use case* adalah teknik untuk meng-*capture* proses bisnis dari prespektif *user*.
    - Aspek statis di-*capture* dalam *use case diagram*
    - Aspek dinamis di-*capture* dalam *interaction diagram*, *statechart diagram* dan *activity diagram*
  - ***Design View***
    - Meliputi *class-class*, *interface*, dan *collaboration* yang mendefinisikan *vocabulary system*
    - Mendukung kebutuhan fungsional *system*
    - Aspek statis di-*capture* dalam *class diagram* dan object diagram
    - Aspek dinamis di-***capture*** dalam interaction diagram, *statechart diagram* dan *activity diagram*
  - ***Process View***
    - Meliputi *thread* dan pendefinisian proses-proses *concurency* dan *syncronization*
    - Menunjukkan *performance*, *scalability* dan *throughput*

- Aspek statis dan dinamis di-*capture* dengan *design view*, tetapi lebih menekankan pada *active class*

- ***Implementation View***
  - Meliputi komponen dan *file* yang digunakan untuk menghimpun dan me-*release system physic*
  - Menunjukkan *configuration management*
  - Aspek statis di-*capture* dalam *component diagram*

    Aspek dinamis di-*capture* dalam *interaction diagram*, *statechart diagram* dan *activity diagram*

- ***Deployment View***
  - Meliputi node yang membentuk topologi *hardware system*
  - Menunjukkan pendistribusian, *delivery*, dan pengistallan
  - Aspek statis di-*capture* dalam *deployment diagram*
  - Aspek dinamis di-*capture* dalam *interaction diagram, statechart diagram, activity diagram*

Definisi design oleh IEEE6 10.12-90 adalah sebagai berikut : “proses pendefinisian arsitektur, komponen, *interface* dan karakteristik lain dari sistem atau komponen” dan “ hasil dari proses itu”. Di tampilkan sebagai proses, *software design* adalah aktivitas terus menerus dari *software engineering* yang mana *software requirements* dianalisa dalam rangka untuk menghasilkan deskripsi dari struktur *internal software* yang berperan sebagai basis untuk konstruksinya. Lebih pastinya, sebuah *softaware design* (hasilnya) harus dapat mendeskripsikan *arsitektur software*. Karenanya, bagaimana *software* dipecah dan disusun menjadi komponen-komponen, dan tampilan antara komponen-komponen tersebut, harus juga dapat mendeskripsikan komponen pada tingkatan detil yang menyediakan konstruksi mereka.

Software design memainkan peranan penting dalam membangun *software*. *Software design* mengijinkan *software engineers* untuk membuat beberapa model yang membentuk sejenis blueprint dari solusi menjadi implementasi.

- **Aktivitas *Software design***

Dalam daftar standar *software life cycle process* seperti pada *Software Life Cycle Processes, software design* terdiri atas dua aktivitas yang sangat sesuai antara *software requirements analysis* dan *software construction*:

***Software architectural design* (sering disebut *top level design*):**

- Menggambarkan software toplevel structure dan mengorganisasi dan mengidentifikasi berbagai komponen.

Software detailed design:

- Menggambarkan tiap komponen secara cukup mengijinkan untuk konstruksinya.

- **General Concepts design**

Software bukan satu-satunya media yang melibatkan desain. Dalam pemaham secara umum, kita dapat melihat desain sebagai bentuk pemecahan masalah. Sebagai contoh, kita mengambil konsep dari masalah yang tidak mempunyai solusi nyata, sangat menarik sebagai bagian untuk memahami batasan dari desain. Sejumlah ide dan konsep lain juga menarik untuk memahami desain dalam pemahaman umum: tujuan, batasan, alternatif, representasi dan solusi.

- ***Software Design Process***

Software design secara umum terdiri atas proses dua langkah:

- ***Architectural Design***

  Architectural design mendeskripsikan bagaimana *software* dipecah dan disusun menjadi beberapa komponen (*the software architecture*)

- ***Detailed Design***

  Detailed design mendeskripsikan perilaku khusus komponen tersebut. Hasil dari proses tersebut merupakan kumpulan dari model-model dan artefak yang merekam keputusan utama yang telah diambil

- ***Enabling Techniques***

1. Prinsip dari *Software Design*, juga disebut dengan teknik penyediaan, adalah ide utama berdasarkan pada berbagai pendekatan dan konsep yang berbeda dari *software design*.
2. Macam *Enabling Techniques* sebagai berikut :

- *Abstraction*
- *Coupling and cohesion*
- *Decomposition and modularization*
- *Encapsulation/information hiding*
- *Separation of interface and implementation*
- *Sufficiency, completeness and primitiveness*

- ***Abstraction***

- *Abstraction* adalah karakteristik dasar dari sebuah entitas yang membedakan entitas tersebut dari entitas yang lain
- *Abstraction* mendefinisikan batasan dalam pandangan *viewer*
- *Abstraction* bukanlah pembuktian nyata, hanya menunjukkan intisari / pokok dari sesuatu

- ***Coupling and cohesion***

Coupling didefinisikan sebagai kekuatan hubungan antara *module*, sementara *cohesion* didefinisikan bagaimana elemen-elemen membuat modul tersebut saling berkaitan.

- ***Decomposition modularization***

Pendekomposisian dan pemodularisasian *software* besar menjadi sejumlah *software* independen yang lebih kecil, biasanya dengan tujuan untuk

menempatkan fungsionalitas dan responsibilitas pada komponen yang berbeda.

- ***Encapsulation***

  Encapsulation adalah menyembunyikan implementasi dari client, sehingga client hanya tergantung pada interface.

- ***The number of key issues crosscutting***

  - *Concurrency*
    Bagaimana *software* dapat membedakan proses, *task*, *threads*, *synchronisasi* dan *scheduling*

  - *Control and handling of events*
    Bagaimana sebuah *software* dapat mengatur data dan flow *control*

  - *Distribtions of components*
    Bagaimana sebuah *software* dapat didistribusikan dan semua komponen saling berkomunikasi

- ***The number of key issues crosscutting***

  - *Error and Exception handling and Fault tolerance*
    Bagaimana sebuah *software* dapat mengenali sebuah *error* dan mengetahui bagaimana cara mengatasinya

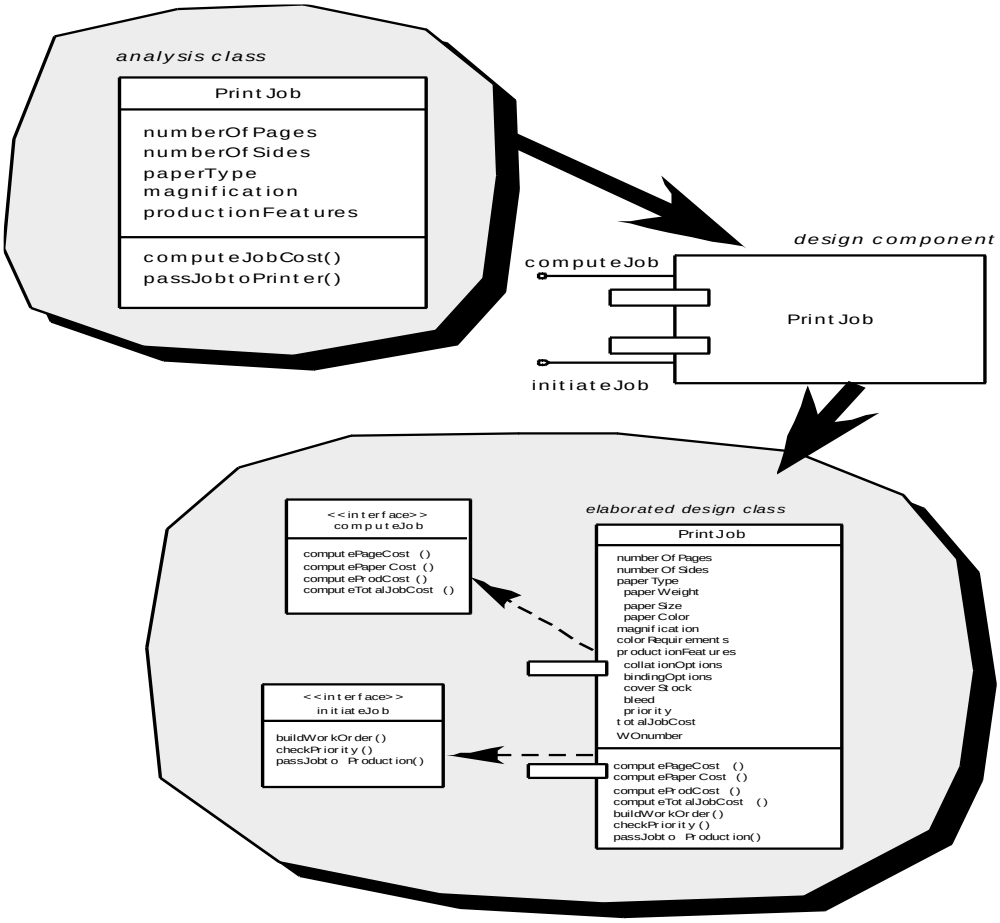
  - *Interaction and presentation*
    Bagaimana sebuah *software* dapat berinteraksi dengan *user* dan dapat menampilkan keinginan *user*

  - *Data persistence*
    Seberapa lama data akan disimpan

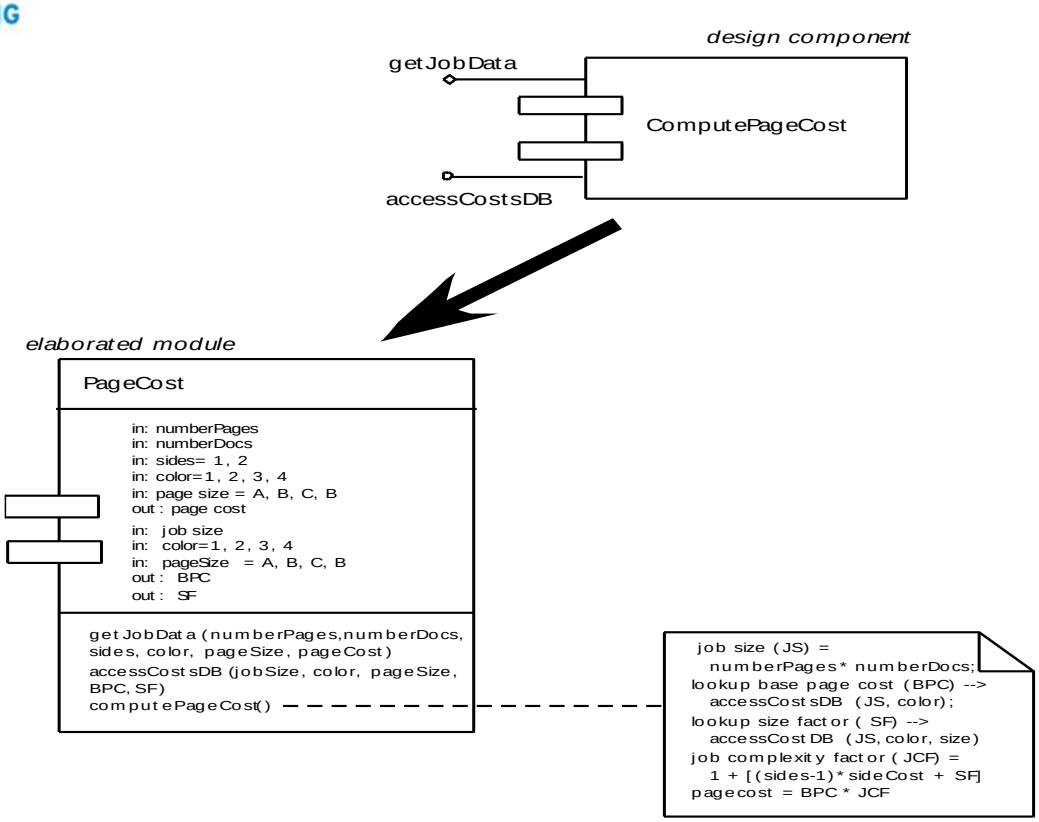
2. *Component-Level Design*

OMG *Unified Modeling Language Specification* mendefinisikan komponen sebagai bagian yang modular, dapat di*deploy* dan dapat digantikan dari sebuah

sistem yang mengenkapsulasi implementasi dan exposes dari sekumpulan interface. Dalam pandangan *object oriented*, sebuah komponen berisi sekumpulan kelas yang saling berkolaborasi.



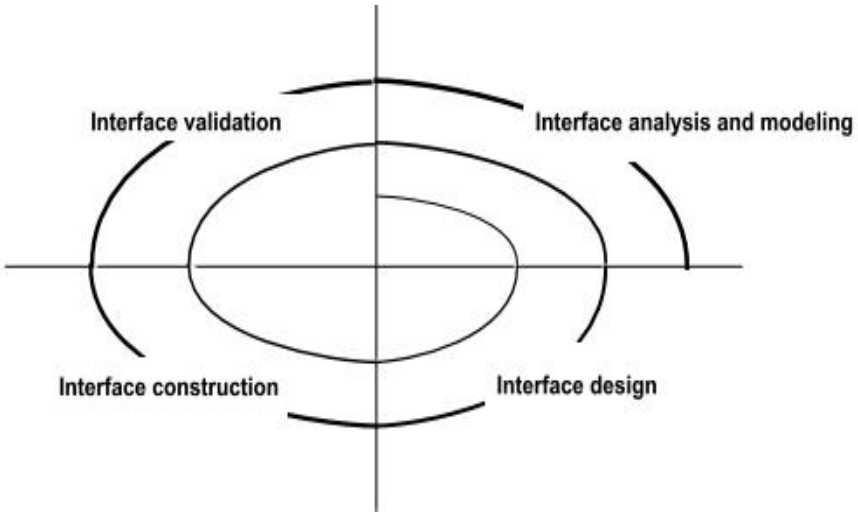
Dalam sudut pandang konvensional, komponen berisi logika pemrosesan, struktur data internal yang dibutuhkan untuk mengimplementasi logika pemrosesan, dan *interface* yang memungkinkan komponen untuk diminta dan data dilewatkan.



3. *User Interface Design*

Berikut ini adalah beberapa model perancangan user interface:

- Model *user*, sebuah profile dari semua *end user* dari sistem
- Model *design*, sebuah realisasi perancangan terhadap model *user*
- Model mental, gambaran mendari dari user terhadap apa itu *interface*
- Model implementasi, bagaimana *interface* “terlihat dan terasa” dengan dukungan informasi yang menggambarkan sintaks dan semantic *interface*.



4. *Pattern Based Design*

Suatu *design pattern* yang efektif memiliki karakteristik sebagai berikut:

a. Memecahkan masalah

b. Membuktikan konsep yang ada

c. Solusi tidak jelas

d. Menjelaskan hubungan yang ada

e. Memiliki komponen manusia yang signifikan

Generative pattern menggambarkan aspek yang penting dan berulang dari sebuah sistem, dan kemudian menyediakan cara untuk membangun aspek tersebut di dalam sistem.

5. *WebApp Design*

Kualitas perancangan dari sebuah aplikasi dapat dilihat dari beberapa faktor berikut:

- ***Security*:**
  - Tahan terhadap serangan eksternal

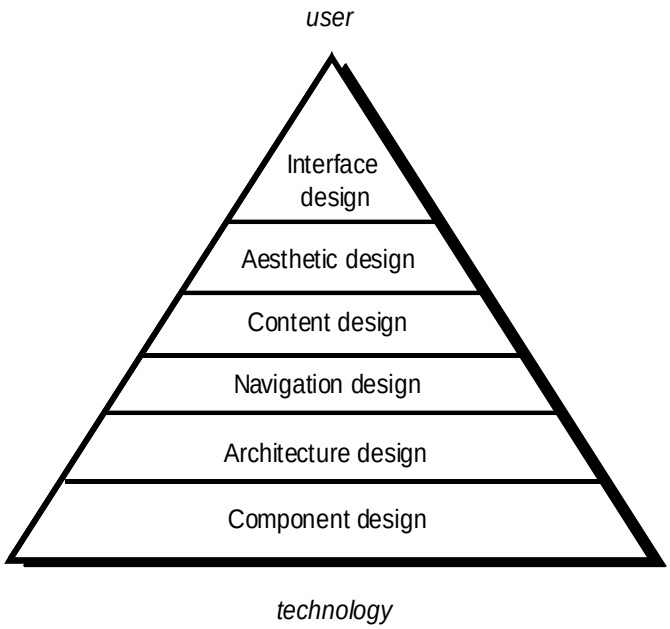
- Menolak akses yang tidak terautorisasi
- Memastikan privasi dari user/customer

- ***Availability***
  - Ukuran dari persentase waktu dimana aplikasi tersedia bagi user untuk digunakan

- ***Scalability***
  - Dapatkah aplikasi web dan sistem interfacenya menghandle beragam signifikasi dari user atau volume transaksi

- ***Time to market***
  - Waktu yang dibutuhkan untuk distribusi dalam pasar.

Berikut ini adalah piramida perancangan untuk aplikasi *web*:



6. *Mobile App Design*

Seperti semua alat *computing*, *platform mobile* dibedakan berdasarkan *software* yang diberikan, kombinasi dari sistem operasi (Android, IoS, dll) dan

bagian kecil dari ribuan aplikasi *mobile* yang menyediakan berbagai fungsionalitas.

Studi Kasus

Pada banyak implementasi software engineering, pada umumnya metode yang dibahas sebelumnya digunakan untuk kondisi pembangunan perangkat lunak yang dibangun dari awal, misalnya Anda ingin membangun:

- Aplikasi penjualan di perusahaan yang spesifik ke perusahaan tertentu
- Aplikasi front-end, misalnya front-end untuk aplikasi perbankan
- Aplikasi marketing yang memiliki fitur yang sangat khusus pada perusahaan tertentu

Beberapa pembuatan atau kustomisasi produk yang dikeluarkan oleh vendor tertentu, misalnya Oracle, SAP, atau IBM terdapat mekanisme dan metodologi tersendiri yang dikeluarkan oleh mereka. Metodologi tersebut dapat digabungkan dengan beberapa konsep desain engineering yang saat ini kita pelajari.

DAFTAR PUSTAKA

- Introduction to software quality assurance,, http://www.youtube.com/watch?v=5_cTi5xBlYg
- Software reliability , http://www.youtube.com/watch?v=wv51aF_qODA
- Lean Six Sigma and IEEE standards for better software engineering,, http://www.youtube.com/watch?v=oCkPD5YvWqw